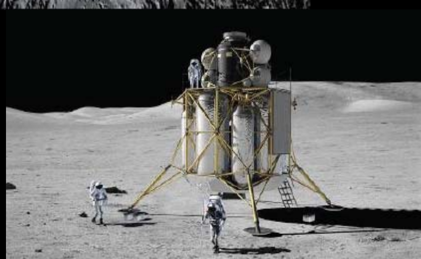
Alternative Software Architecture Development Approaches for Lunar Surface Systems

Presented to the US Chamber of Commerce Programmatic Workshop on NASA Lunar Surface Systems

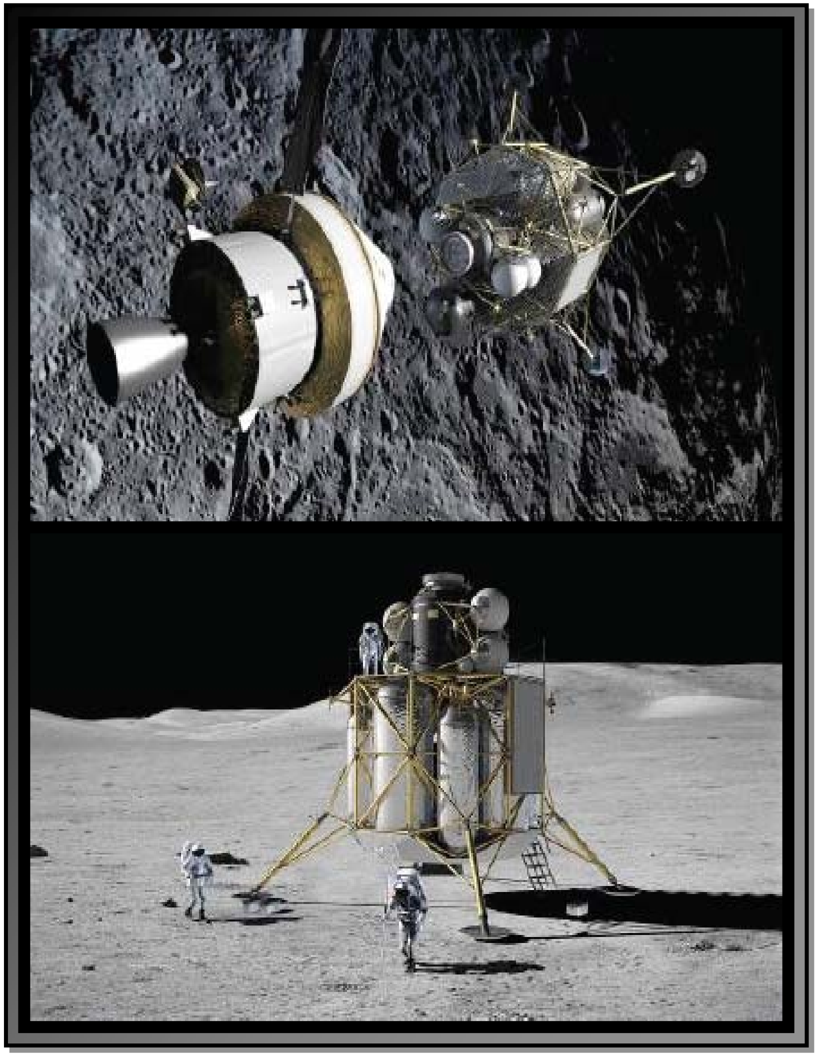


Roscoe Ferguson
Graham O’Neil
26 February 2009

United Space Alliance

Lunar Surface Systems Concept Study Topic 5

- **Objective is to provide NASA with identified alternate software development approaches and architecture for the LSS to increase software reliability and performance, while decreasing the development and maintenance costs of that software.**
- **Define Figures of Merit (FOM) specifying significant contributors to development and maintenance costs.**
- **Evaluate approaches in regards to effectiveness against significant cost drivers.**
- **Provide example of architecture as applied to LSS.**

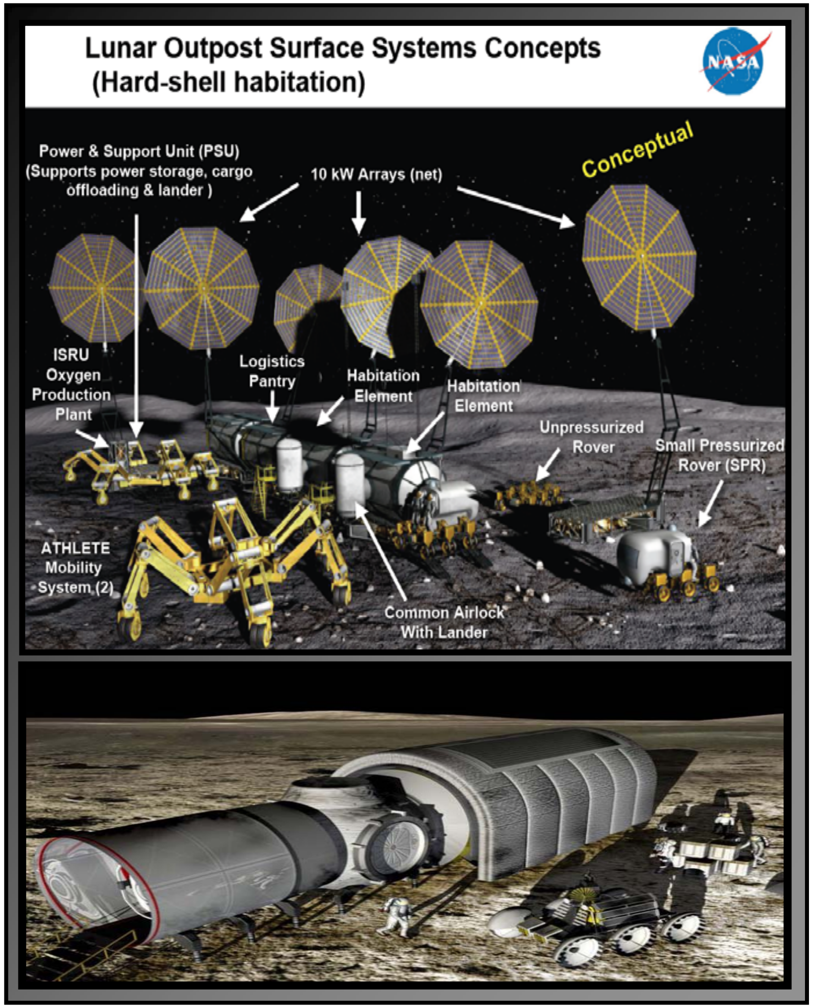


Topics

- Lunar Mission from a Software Viewpoint
- Cost Drivers and Figures of Merit
- Development Approaches
- Software Architecture and Design
- Comparison and Results

Lunar Surface System Elements

- The LSS will consist of a fleet of systems including crew habitats, rovers, power systems, oxygen production plants, and laboratory systems.
- Crew habitats will support a crew of 4 for 180 days on the lunar surface.
- Rovers will be operated autonomously or by the crew. There will be pressurized roving systems that can travel for hundreds of kilometers.
- Power systems will produce at least 35 kW of net power production and storage for eclipse periods.
- Oxygen production plants will produce oxygen at a rate of 1 mT per year.
- Laboratory systems will provide laboratories and instruments to meet exploration and science objectives.

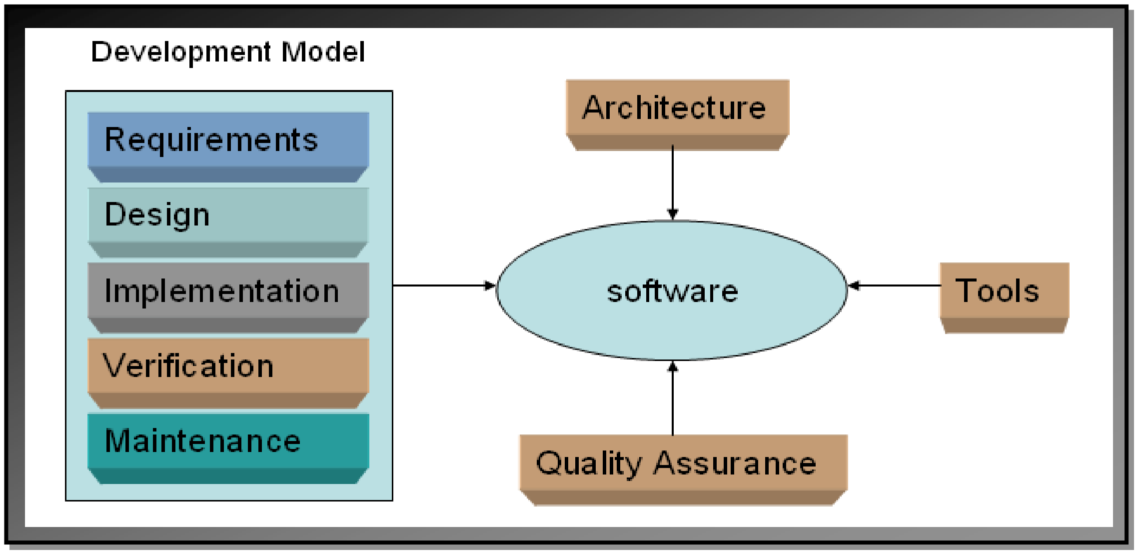


LSS Software Challenges

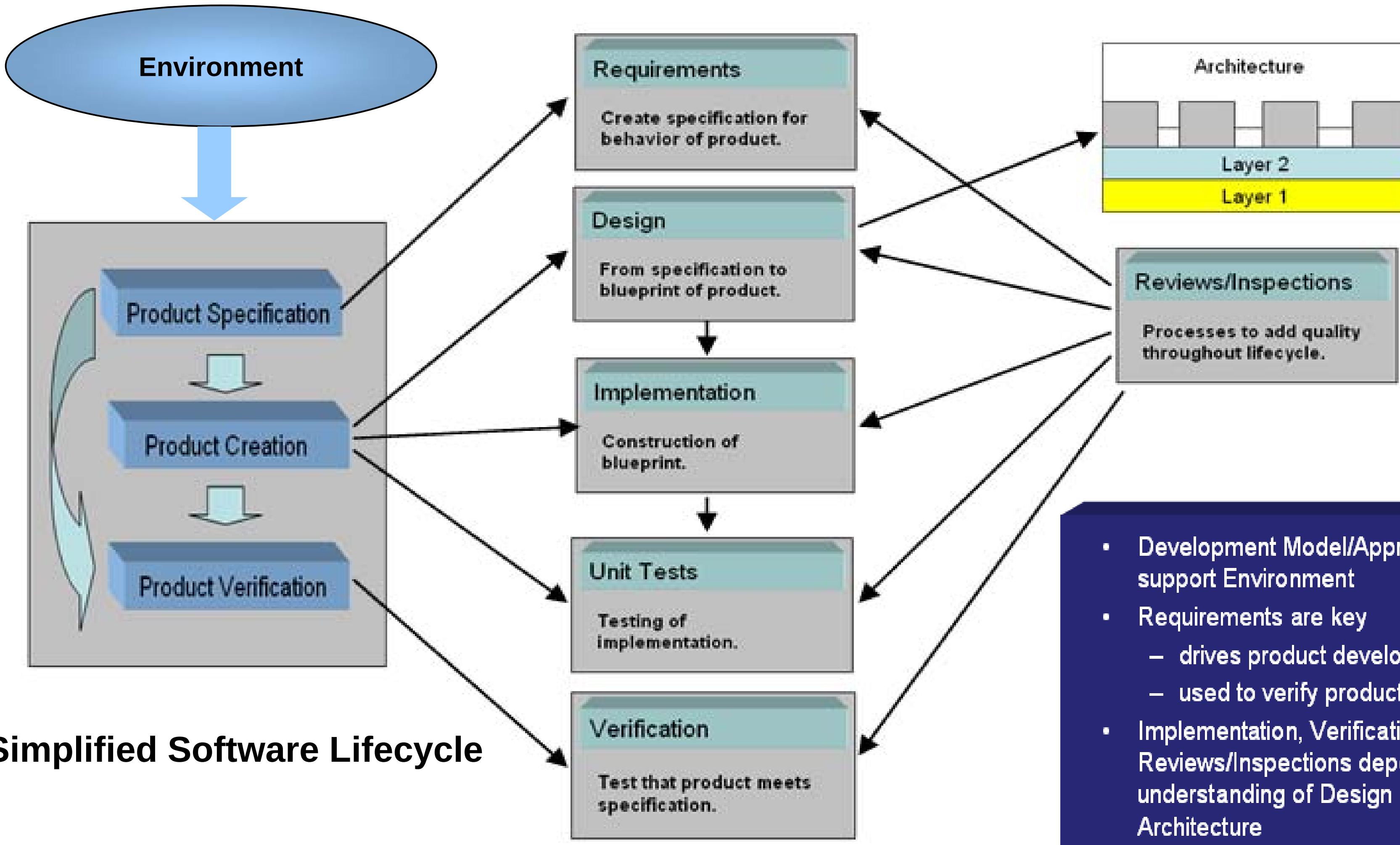
- Distributed Cooperation
- Distributed Failure Management
- C3I Compliance
- Varying Levels of Fault Tolerance
- Remote Operation
- Autonomous Operations
- Transitions from Dormancy to Reconstitution
- Integration of New Software on Non-Interference Basis
- Accommodation and Integration of International Partner Software Systems
- All complicated by operations in the Lunar environment.

Software Architecture and Development Models

- A software development process is a methodology used to control the development of a software product.
- The Object Management Group (OMG) defines software architecture as the specification of the parts and connectors of the system and the rules for the interaction of the parts using the connectors.
- Software development processes and software architectures can have a profound effect on software development and maintenance cost.



Software Architecture and Development Models



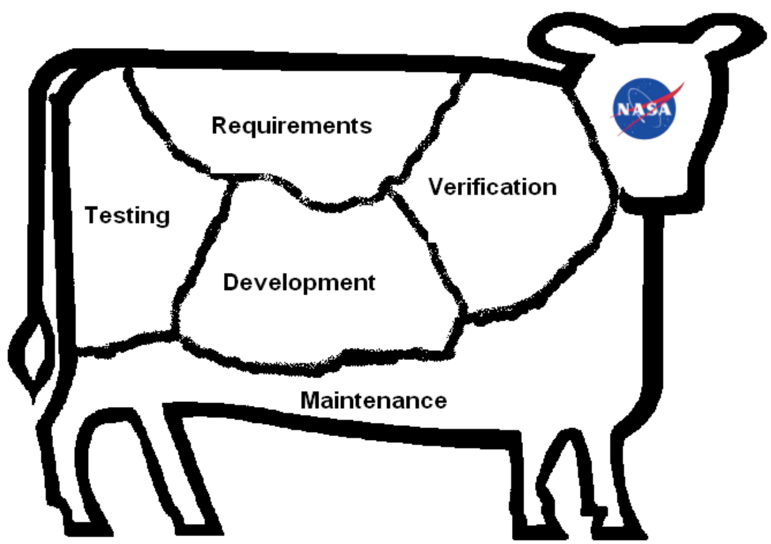
Simplified Software Lifecycle

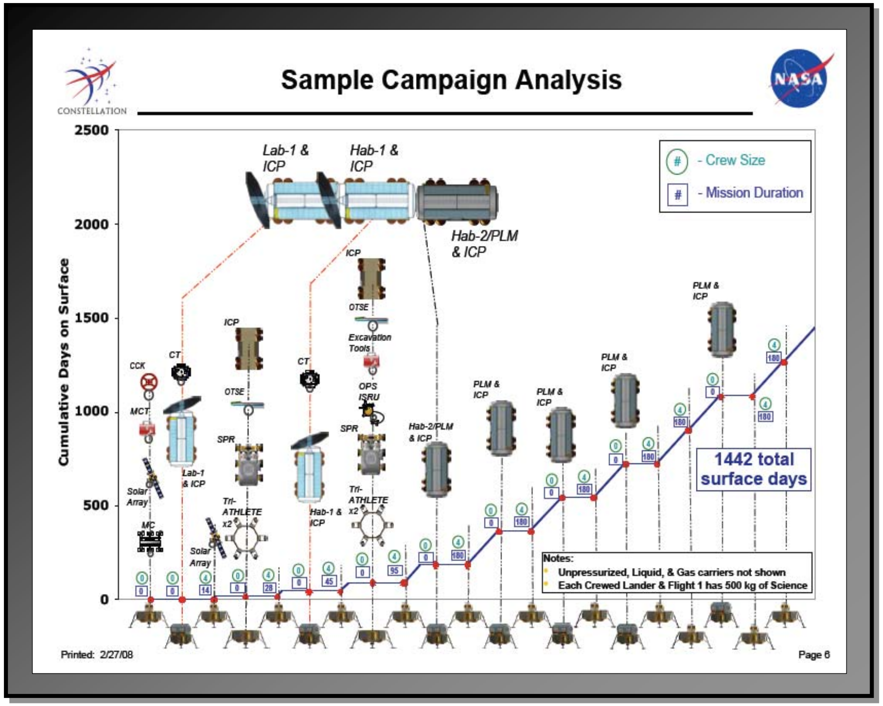
- Development Model/Approach must support Environment
- Requirements are key
  - drives product development
  - used to verify product
- Implementation, Verification, Reviews/Inspections depend on understanding of Design and Architecture

Cost Drivers and Figures of Merit

Identified Cost Drivers/Figures of Merit (FOM)

- **Cost = labor*time**
- **Cost drivers associated with:**
  - *Software Development Approaches*
  - *Software Architecture*
- **Cost drivers attributed to time, labor or both**





United Space Alliance

Cost Drivers - Software Development Approaches

Human Resource Management

Efficiency in reducing idle time of human resources during software lifecycle. Increased productivity can reduce development time. (time)

End of Lifecycle Defect Rate

Rate of major errors found late in the lifecycle. Errors typically found late in the lifecycle increase cost. (time, labor)

Requirements Scope Production

Effectiveness of development approach to support "buy by the yard". Provides ability to produce reduced scope systems to be augmented in the future. Reduces total loss of investment. (time, labor)

Requirements to Product Alignment

Efficiency to prevent divergence between requirements and implementation during the lifecycle. Divergence results in rework. (time, labor)

All Lifecycle Steps

Lifecycle Steps

Requirements

Create specification for behavior of product.

Design

From specification to blueprint of product.

Implementation

Construction of blueprint.

Unit Tests

Testing of implementation.

Verification

Test that product meets specification.

Requirements Convergence

Efficiency to converge requirements. (time)

Requirements Testability

Degree of testability of requirements to support verification. Vague or ambiguous requirements result in rework (time, labor)

Change Efficiency

Efficiency to make changes during the software lifecycle. (time, labor)

Software Lifecycle Artifact Automation

Level of automation in the production of lifecycle artifacts. Reduces the traditional manual effort required to support lifecycle artifacts. (time, labor)

Software Lifecycle Tool Support

Level analysis aids, development process automation, and integration. Automation can reduce manual efforts. (time, labor)

Tools

Cost Drivers - Software Architecture/Design

Design Abstraction
Level that represents the rendering of concepts to realize a design. Increases the understanding of a system. (time, labor)

Design Pattern Consistency
Level consistency when implementing similar design concepts. Consistent design patterns increases the understanding of a system (time, labor)

Encapsulation
Packaging of behavior into containers. It provides conceptual and physical independence. Enhances the ability to adapt and understand changing the system. (time, labor)

Scalability
Property of system to gracefully handle growing amounts of scope. Enhances the changeability of a system. (time, labor)

Reuse Factor
The amount of reuse in software product. Reuse reduces reimplementation of common concepts and increases reliability be the reuse of proven software components in a system (time, labor)

Lifecycle Steps

Requirements
Create specification for behavior of product.

Design
From specification to blueprint of product.

Implementation
Construction of blueprint.

Unit Tests
Testing of implementation.

Verification
Test that product meets specification.

Reviews/Inspections
Processes to add quality throughout lifecycle.

Modularity
The degree of the separation of concerns by enforcing logical boundaries between components using well defined interfaces. Enhances understanding and changeability of a system (time, labor)

Technology Independence
The ability to evolve domain logic to newer technologies. Software is usually written for a specific application that makes a large investment in domain logic. (time, labor)

Software Partitioning
Property to physically isolate software areas or groups. Improves reliability and changeability of system. Creates isolated verification regions of software. (time, labor)

Fault Tolerance Level
Property that indicates how many faults a system can encounter and continue proper operation after failure. Cost typically increases the more faults a system is tolerant to before failure. (time, labor)

Fault Tolerance Approach
The approach to implement fault tolerance. Complex in nature and can increase complexity of system. Can be implemented in software or hardware. (time, labor)

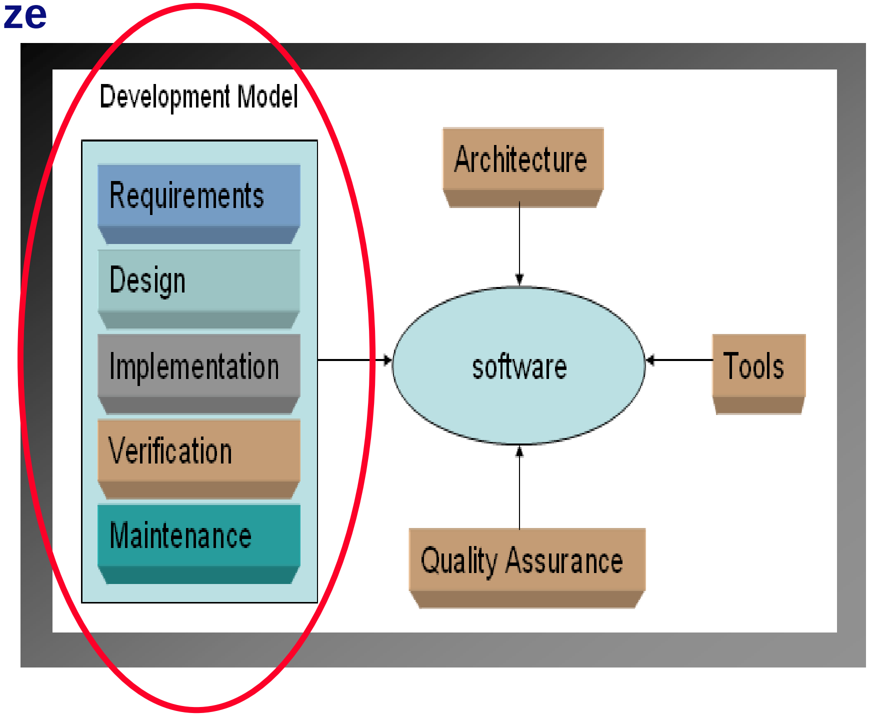
All except Requirements

All except Reviews/Inspections

Development Approaches

Identified Development Approaches

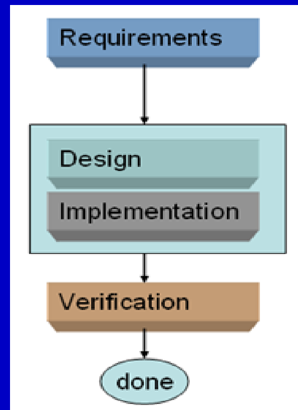
- **There are various models or approaches for software development, but all can be broken down into the steps of Requirements, Design, Implementation, Verification, and Maintenance.**
- **Each model provides a philosophy to realize each step and the relationships between them.**
- **Identified approaches are:**
  - *Academic Waterfall Models*
  - *Spiral Model*
  - *Iterative Model*
  - *Agile Methods*



Identified Development Approaches

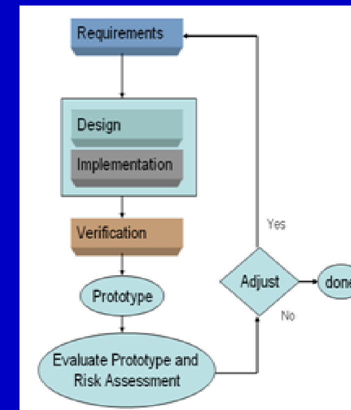
Academic Waterfalls

- Well defined processes
- Up-front requirements and planning
- Steps performed serially (except modified approach)
- Exponential cost curve over time
- Relies on artifacts (documentation)
- Little or no feedback from experience



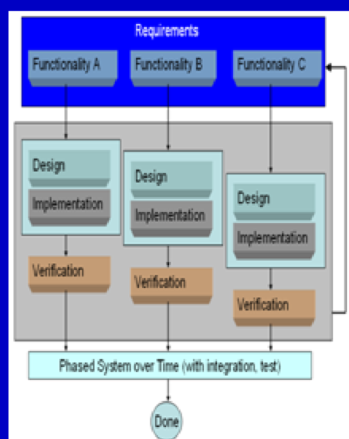
Spiral

- Like Waterfall, except each step is ended with a prototyping effort and risk assessment.
- Prototype lets users determine if project is on track, should be sent back to prior steps, or should be ended.
- Includes Risk Planning as part of process.



Iterative

- Requirements partitioned into prioritized functionality groups (subsystems) with clean interfaces.
- Each iteration consists of all Waterfall steps.
- Each iteration only addresses one set of partitioned functionality.
- Each iteration can be a full production system.
- Feedback from iterations



Agile Methods

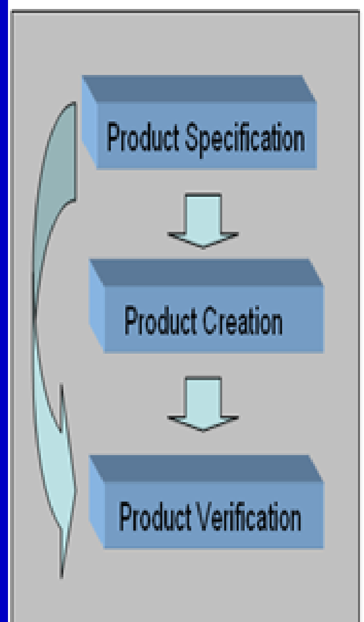
"QB audible at the line"

- Minimal up-front planning with broad up-front requirements.
- Adaptive to change based on environment.
- Short time frame iterations of full development cycles (requirements – testing) resulting in working software.
- Features frequent communications over documentation via team collaboration with customer involvement.
- Working software is primary measure of progress vs. artifacts.
- Relies on techniques for quality and productivity (continuous integration, test automation, pair programming, test driven development).

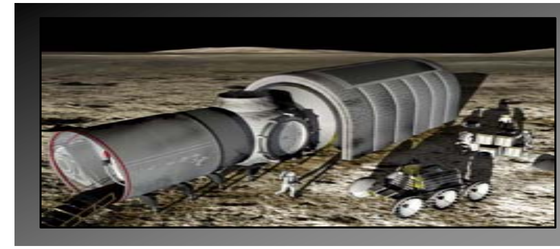
LSS and Development Approaches

All have useful features. Each taken into consideration poses the questions:

- Complete specification (requirements) vs. modularized evolved specification?
- How much feedback from experience (trial and error)?
- Inspection vs. applied techniques for quality improvement?
- How much and what levels of collaboration?



Answers depend on the software criticality, complexity and the environment.



- **LSS has the potential to be developed and maintained in a dynamic and constrained environment.**
- **Will have multiple elements of varying criticality and complexity.**
- **Possible approach:**
  - *extract "best practices"*
  - *apply them as needed in combinations for LSS element based on criticality and complexity.*

Development Approach Best Practices and Cost Drivers

- **Modularized Requirements with Priority**
  - *Allows scope to be broken up as required in modules*
  - *Modules can be parts, subsystem, or entire system (for simple systems)*
  - *Can occur in parallel or phased*
- **Provide feedback to requirements module via working software**
  - *Trial and error to reduce risk*
- **Build verification test side by side with requirements (test driven approach)**
  - *Ensures requirements are testable*
- **Build unit tests before development (test driven approach)**
  - *Provides quality first mentality for development*
- **Use Pair Implementation where two developers work together at one machine. A driver enters the implementation and another critiques it. Roles are periodically switched.**
  - *Claimed to increase productivity*
  - *High quality code (15% fewer defects) in about half the time (58%). Williams, L., Kessler, R., Cunningham, W., & Jeffries, R. Strengthening the case for pair programming. IEEE Software, 17(3), July/August 2000*

Addressed Development Process Cost Drivers

Development Approach Best Practices and Cost Drivers

- **Tight iteration durations and continuous testing**
  - *Forces productivity*
  - *Early and frequent error detection*
  - *Increases feedback rate*
  - *Minimizes specification to product divergence*
- **Use working software as progress**
  - *Provides actual measure of progress*
- **Frequent collaboration with customers or stakeholders**
  - *Minimizes project divergence from expectation*
  - *Customer or stakeholder really aware of program state.*
- **Use Inspections/Reviews**
  - *Dependable approach for quality*
  - *Can be more efficient with use of analysis tools and well established software and design practices*

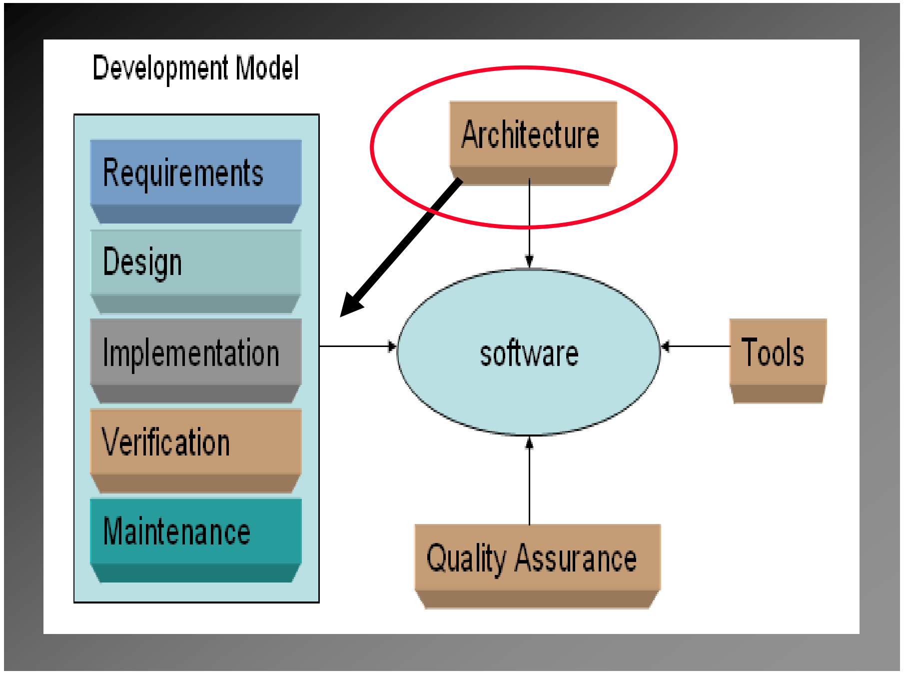
Addressed Development Process Cost Drivers

United Space Alliance

Software Architecture and Design

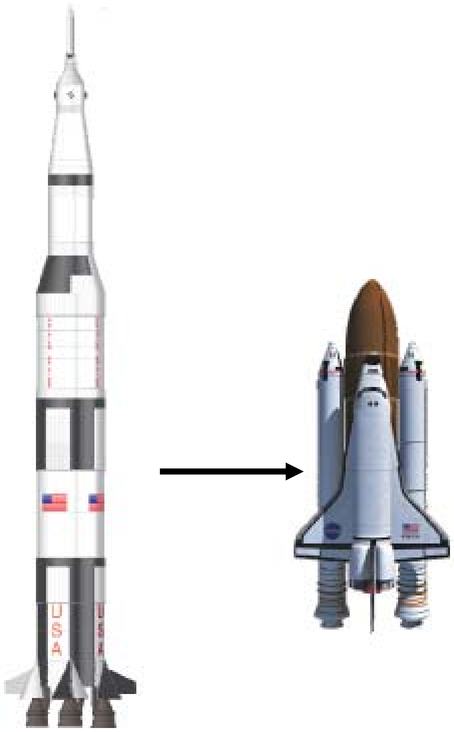
Identified Alternate Software Architectures/Design

- Software Architecture and design decisions have a direct effect on development model lifecycle costs.
- The architecture and design decisions should utilize modern and proven key modern software methods and techniques.
- Should also look for an architecture approach that is designed to support development lifecycle concerns.



Apollo Program to Space Transportation System

- **Assembly language style programming was the status quo.**
- **Belief that high order languages with assembler (code generation) was not usable.**
- **Competition was performed to determine feasibility of modern software practice (high level programming).**



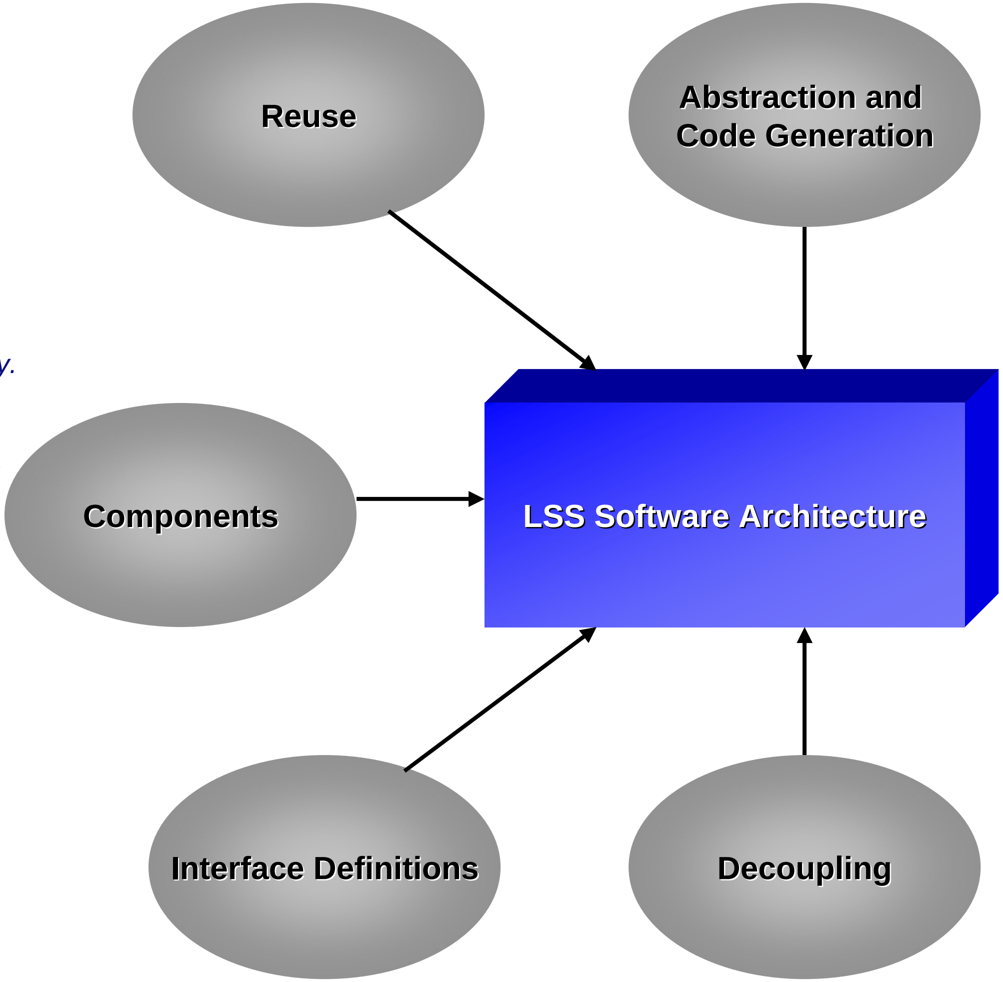
The competition showed that the approximate 10 percent loss in efficiency resulting from the use of the high-order language was insignificant when compared to the advantages of increased programmer productivity, program maintainability, and visibility into the software.

Use of high-level languages coupled with improved development techniques and tools, productivity was doubled over the comparable Apollo development processes.

Higher levels of abstraction and code generation improved the software development and productivity in the 1970’s and should be effective for the transition to the LSS.

Identified Design Decisions

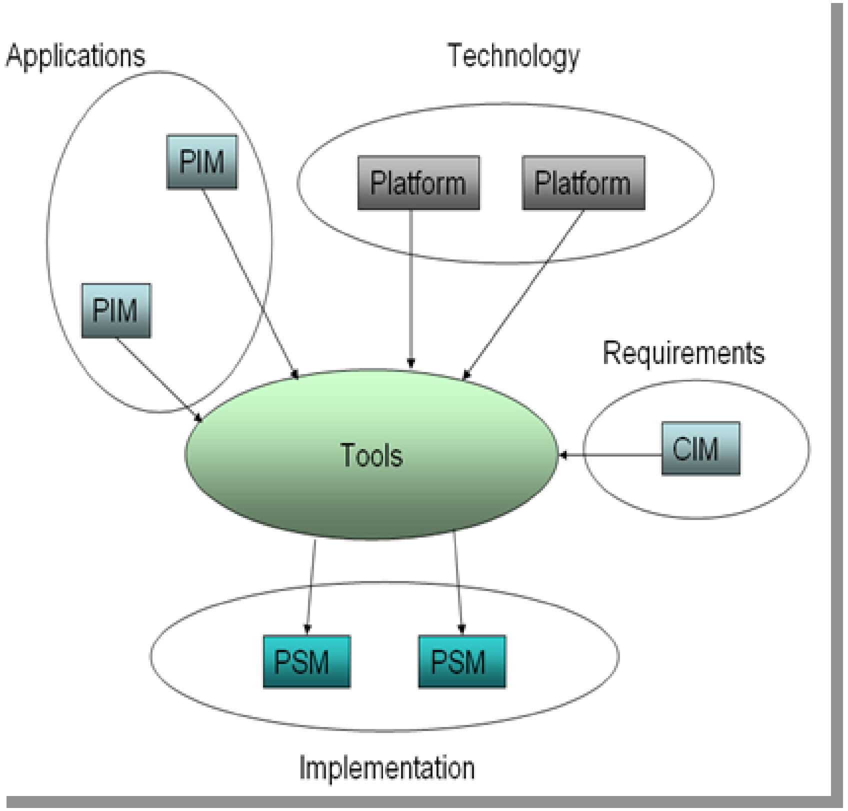
- **Abstraction and Constrained Code Generation**
  - *Increase system understanding*
  - *Provide consistent design patterns*
  - *Build software like hardware, concepts implemented as combinations of common design patterns and principles*
- **Reuse**
  - *Build common functions and components once for all LSS elements to decrease cost and increase reliability.*
- **Component Based**
  - *Modularize internal system to services with interfaces*
- **Interface Definitions**
  - *For interoperable system to system interaction definitions*
- **Decoupling**
  - *Publish/Subscribe data distribution can improve data accessibility both internally and test facility support.*



Identified Architecture

- **Model Driven Architecture (MDA)**
  - *Model Driven to direct the course of understanding, design, construction, deployment, operation, maintenance, and modification.*
  - *Platform based (layering)*
- **Requirements in Computation Independent Models (CIMs)**
- **Application or domain logic in Platform Independent Models (PIMs)**
- **Implementation and services in Platform Specific Models (PSMs)**
- **Tools provide**
  - *Traceability between CIM, PIM, and PSM.*
  - *Model compilers and supporting artifacts.*
- **Promotes**
  - *Portability*
  - *Interoperability*
  - *Reusability through architecture separation of concerns*

MDA Process



United Space Alliance

Identified Software Architectures/Design and Cost Drivers

- **Design Approaches**
  - *Abstraction and Constrained Code Generation*
  - *Reuse*
  - *Component Based*
  - *Interface Definitions*
  - *Decoupling*
- **Architecture**
  - *Model Driven Architecture (MDA)*

Addressed Architecture/Design Cost Drivers

Software Partitioning

Property to physically isolate software areas or groups. Improves reliability and changeability of system. Creates isolated verification regions of software. (time, labor)

Can be provided in technology such as ARINC 653 RTOS

Fault Tolerance Level

Property that indicates how many faults a system can encounter and continue proper operation after failure. Cost typically increases the more faults a system is tolerant to before failure. (time, labor)

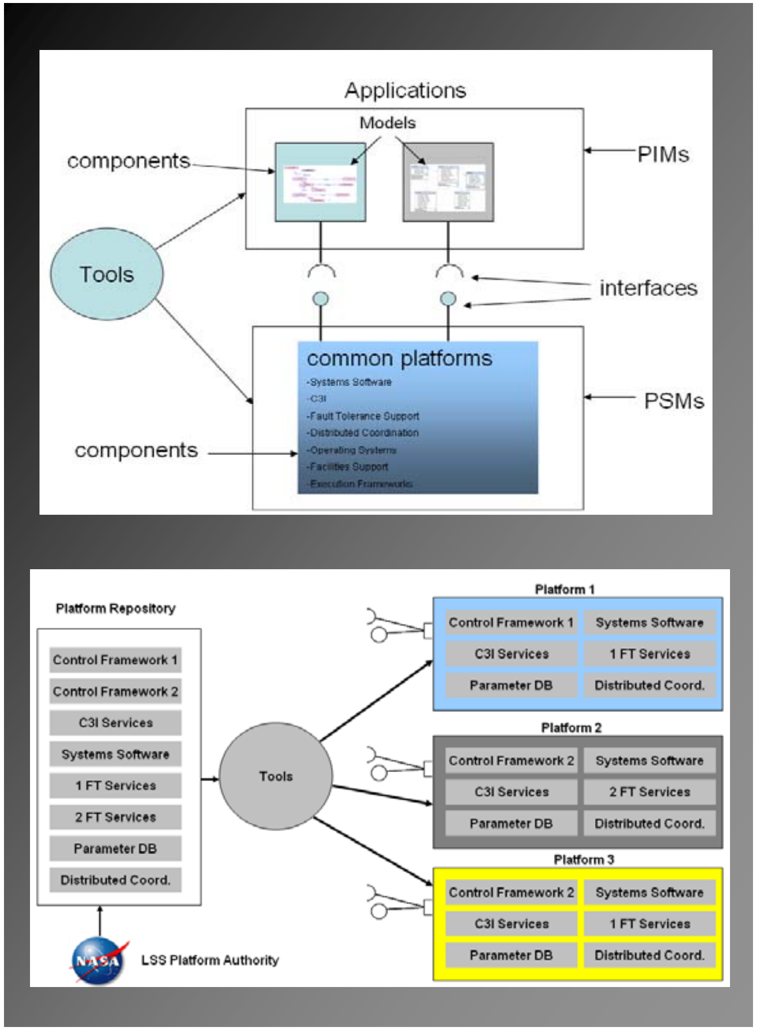
Fault Tolerance Approach

The approach to implement fault tolerance. Complex in nature and can increase complexity of system. Can be implemented in software or hardware. (time, labor)

Independent of MDA and depends on approach

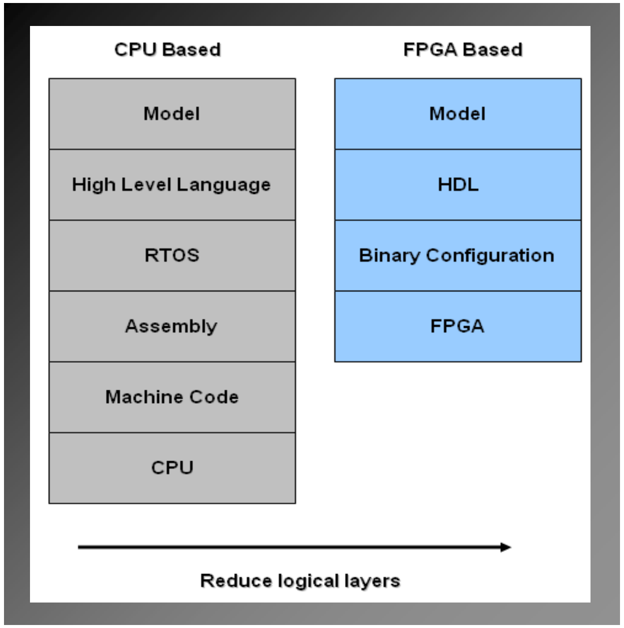
Example of LSS using MDA

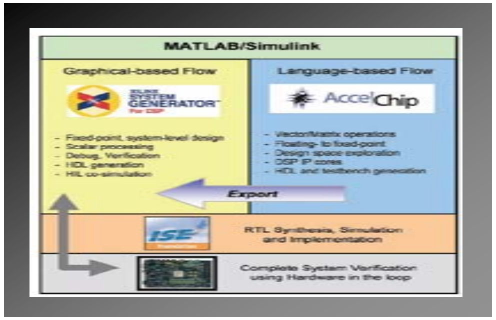
- **Common platform services provided by central authority as form of middleware (PSMs)**
- **Reuse established in PSMs:**
  - *system software*
  - *distributed coordination*
  - *fault tolerance support*
  - *facility support*
  - *C3I*
- **Domain logic implemented in models encapsulated via components and interfaces (PIMs)**
- **Tools can support PIM and PSM**
  - *Model Compilers for PIM (PSM can be hand coded if required)*
  - *Generate PSM based on LSS element needs*



Example of MDA Support of Technology Independence

- **Domain logic implemented as Matlab models in PIM is independent of technology.**
- **Can be generated to support alternate technology such as “Reconfigurable Computing” using SRAM Field Programmable Gate Arrays.**
- **Performance Gains**
  - *control applications implemented directly in hardware execute in parallel*
- **Increased Reliability**
  - *Algorithms implemented using common and strict hardware design patterns*
  - *Removes complex software analysis tasks*



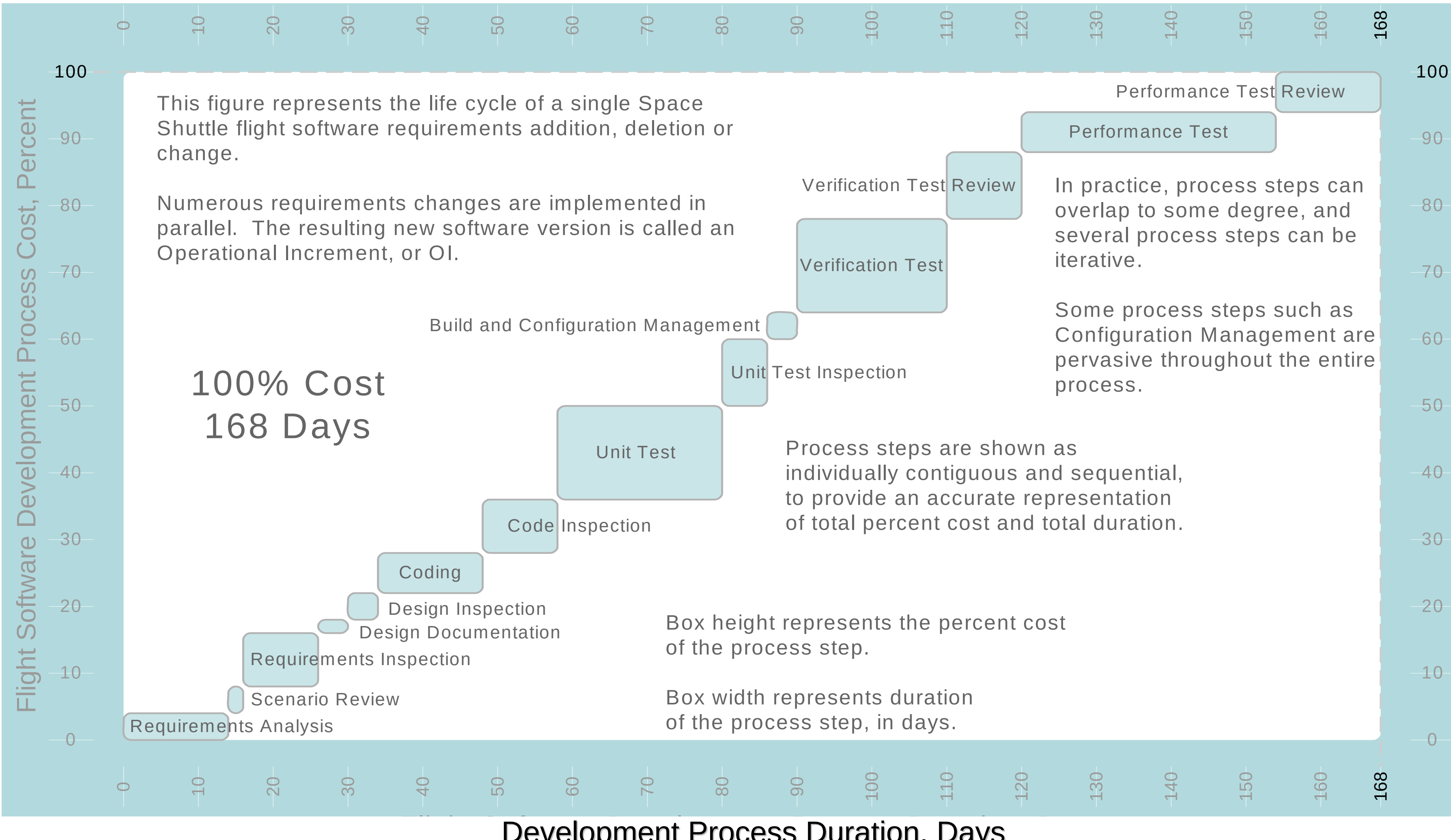


Comparison and Results

Program Evaluations of Improvement Contributions

Contributors\Program	Shuttle	ISS	CEV	QRMS	Nuclear Power
Model Driven Architecture	X		X		
Model Reuse	X	X	X		X
C3I		X	X	X	
Decoupling	X			X	X
Simple Interface	X			X	X

Typical High Maturity Development Process



United Space Alliance

LSS Reuse Example: Basic Parameters

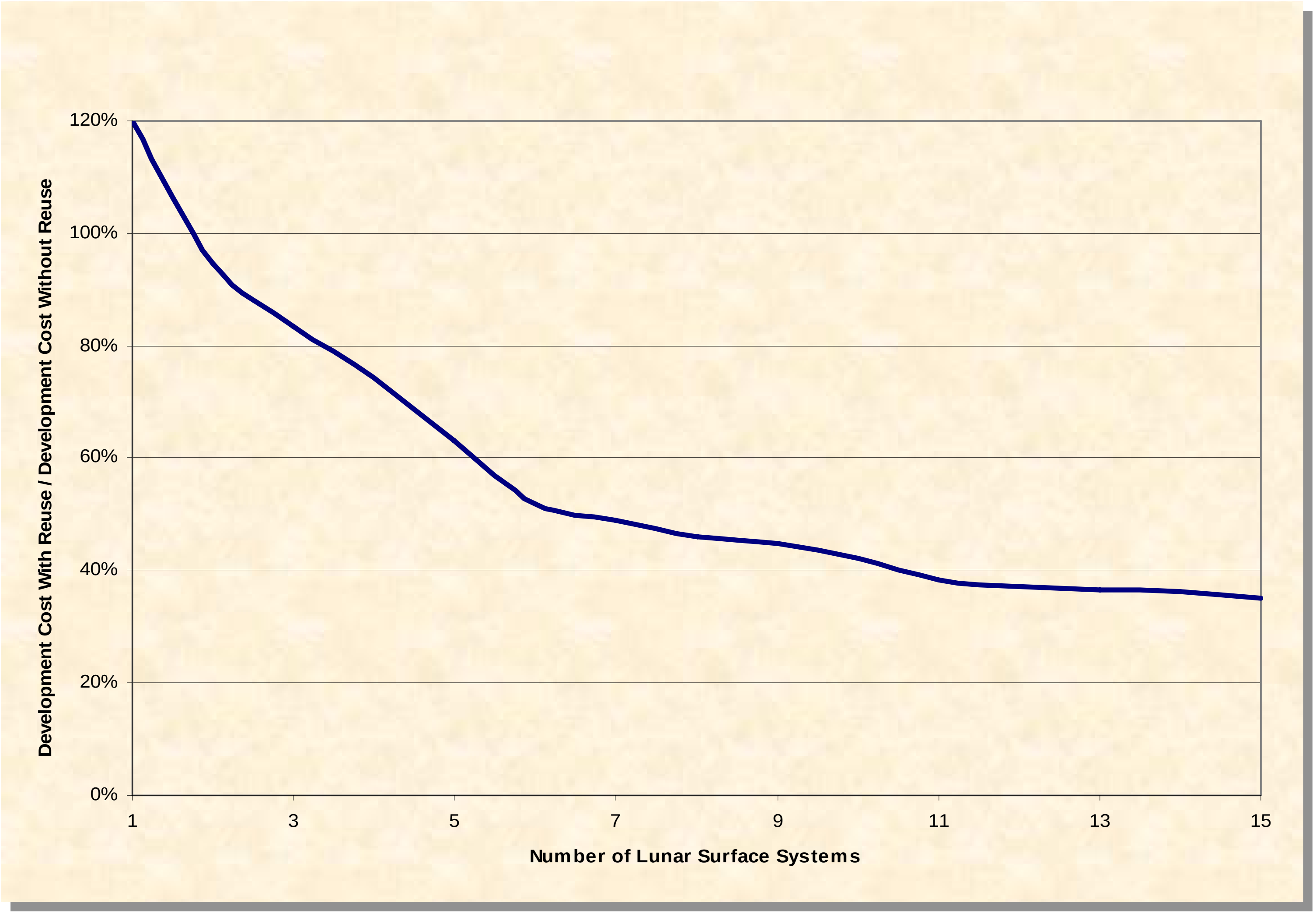
Platform	Source	Total Size	Re-used	New
Habitation System	JSC Habitat Testbed	250	0	250
Oxygen Generation	US Navy	80	70	10
Power System	Commercial	100	80	20
Exploration Science	Estimate	120	90	30
Rovers	INEL	150	140	10

Assume The Following Characteristics…

- **Latent Defect Rate is .1 defect per KSLOC. This would be very good.**
- **Defect Insertion Rate is 2%. This is very good.**
- **Probability a defect leads to a Crit 1 failure; Loss of Crew or Vehicle is 2%**
- **Latent Defect Removal Rate is 15% per year. This is very good.**

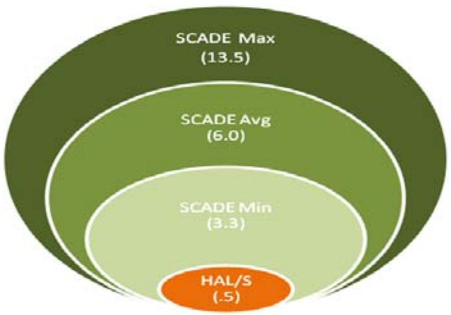
Reuse Provides Additional Safety Margin

Reuse Provides 50% Cost Reduction



Model Driven Architecture Results

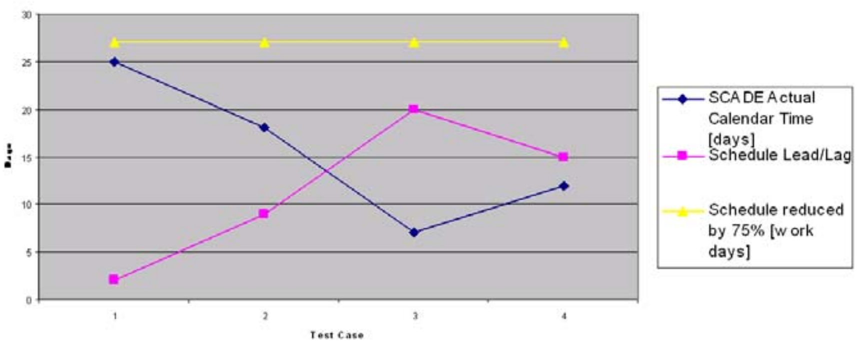
Productivity



Average productivity of SCADE across 4 test cases is an order of magnitude greater than current software development.

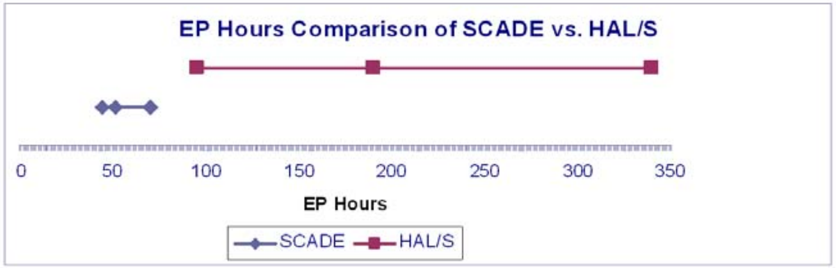
Schedule

Performance Against 75% Reduction Goal



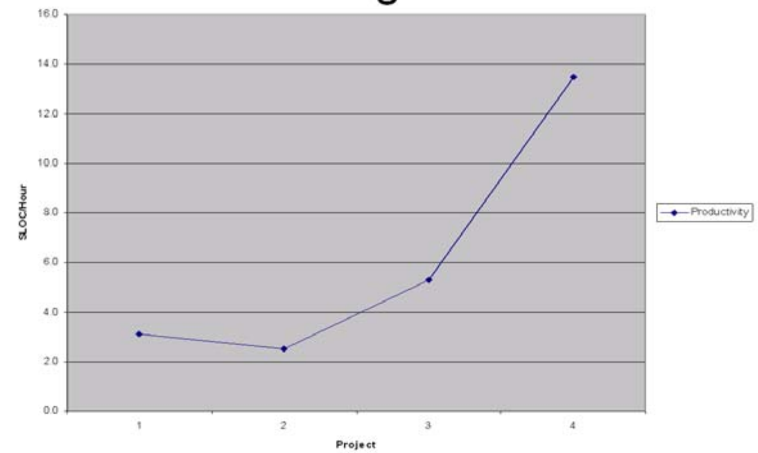
All four test cases were completed within the reduced schedule goal.

Effort



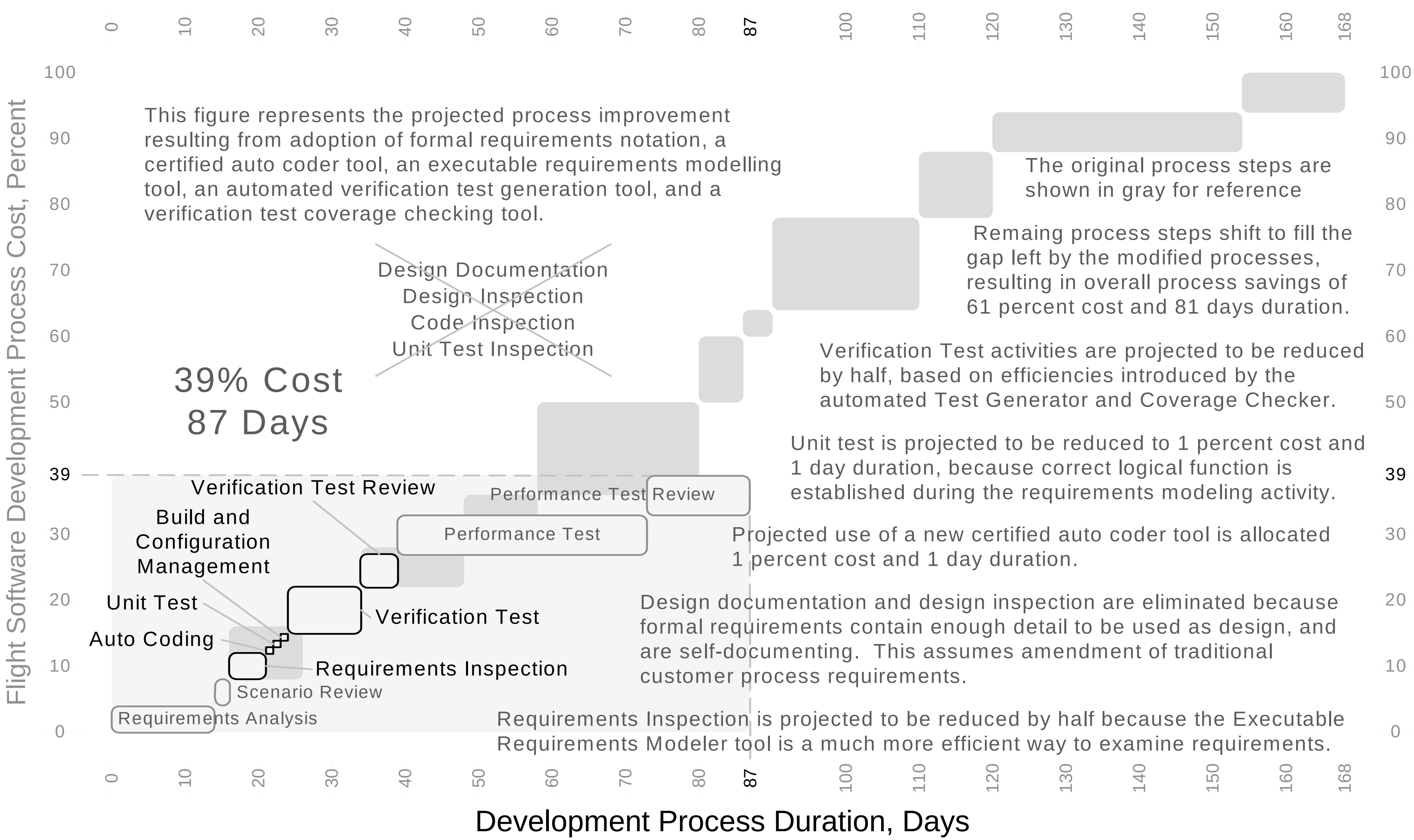
Starting from scratch, SCADE produced zero defect modules with less development and verification effort than standard approaches.

Learning Curve



United Space Alliance

Optimized High Maturity Software Development Process



USA United Space Alliance

Additional Application Areas for LSS Software Solutions

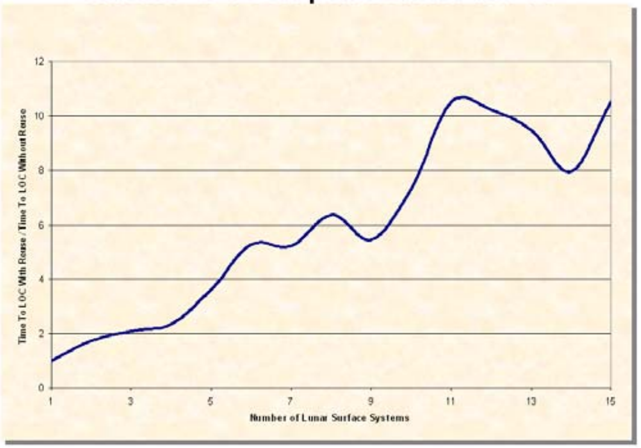
- **Systems with similar characteristics for high reliability, automation of complex actions, driven by data that is dynamic in a dynamic environment.**
  - **Human Medical Systems [both in-vivo and in silico]**
  - **Urban Traffic Management**
  - **Rail Road Control Systems**
  - **Chemical Plants**

Further Studies

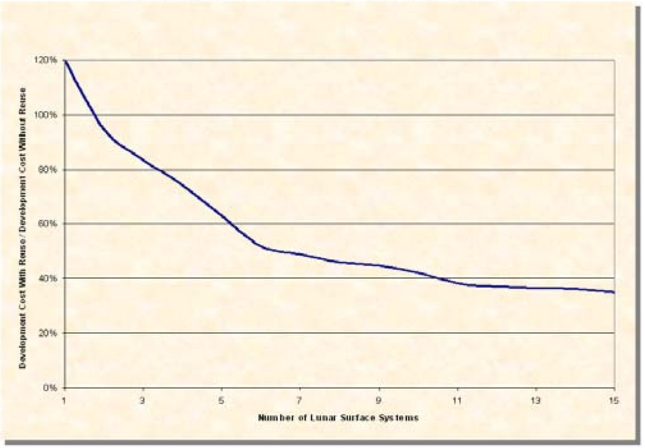
- Reuse case studies to determine barriers and mitigations for improved reuse.
- Integration of Systems and Software in the Operational stage.
- Assessment of Competency in systems as part of IVHMS.
- Interfaces for Integration with International Partners
- Study of approaches for constrained code generation.
- Study of the use of tools to cut cost in the development lifecycle including analysis aids and lifecycle support.
- Requirements and Design tool support for system dormancy and reconstitution functions.

Conclusion

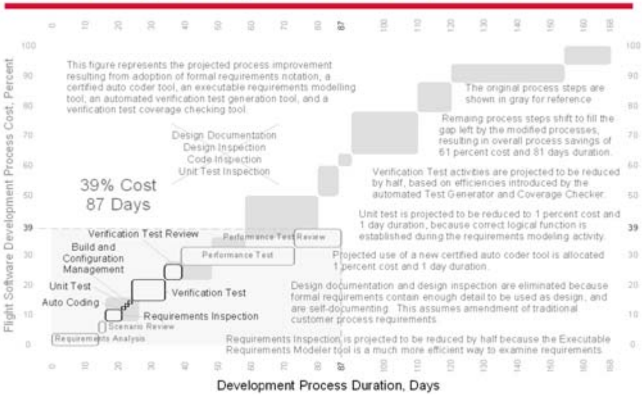
Ratio Of LOC Risk: Non-Reuse Compared to Reuse Development Methods



Ratio Development Cost of Reuse Compared To No Reuse



Optimized High Maturity Software Development Process



1. **Reliability, quality and safety goals can be met at reduced cost and effort of current human space flight systems.**
2. **Most important contributors to cost reduction for high reliability systems are already being partially used.**
3. **Developing organization must focus on standardization, inspection, test, and select the appropriate development approach for the system.**